FIAP ON

DESENVOLVIMENTO DE APPS – PARTE 1 (ANDROID)

PERSISTÊNCIA **DE DADOS** LOCAIS

GUSTAVO CALIXTO

5

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

5 PERSISTÊNCIA DE DADOS LOCAIS

A persistência de dados em aplicativos para dispositivos móveis permite ao desenvolvedor armazenar dados localmente para viabilizar, por exemplo, informações personalizadas ao usuário (como uma saudação com o nome da pessoa e tratamento), configurações locais, perfis de usuário e até mesmo operar com funcionalidades off-line como uma forma de redundância a uma funcionalidade on-line, ou seja, que depende da conectividade de rede.

Por sua vez, na plataforma de desenvolvimento Android, existem três maneiras conhecidas de como realizar a persistência de dados, como segue:

- SharedPreferences: a classe SharedPreferences permite o armazenamento de dados por meio de pares “chave e valor”, similar ao uso de hashmaps em Java.
- Uso de arquivos: é permitida a criação de arquivos que podem ser lidos e gravados por meio do gerenciamento em código-fonte.
- Uso de banco de dados SQLite: o Android possui um banco de dados SQLite nativo para uso Ele é um banco de dados relacional com padrão SQL, que tem por característica ser um *serverless*, ou seja, não necessitar de um SGDB (Sistema de Gerenciamento de Banco de Dados) para a sustentação. Internamente, no dispositivo móvel, é criado um arquivo em que possuirá todas as tabelas criadas e restrições (chave primária, chave estrangeira, nularidade, unicidade, etc.).

Nas seções a seguir, serão apresentados exemplos práticos do uso de cada uma das maneiras de persistir dados na plataforma Android.

5.1 SharedPreferences

Como já comentado anteriormente, a caraterística do SharedPreferences é o armazenamento de dados por formato “chave e valor”. A seguir, é apresentado um projeto no qual a saudação preferida por um usuário é gravada e exibida na tela seguinte. O exemplo utilizará duas Activities chamadas de MainActivity (já criada pelo

Android Studio na criação do projeto) e outra chamada de SaudacaoActivity, que exibirá a saudação gravada pelo usuário.

Com o projeto criado no Android Studio com uma Empty Activity e em linguagem de programação Kotlin, configure a MainActivity para projetarmos o layout em FrameLayout (ou também o tipo de layout da sua escolha) para adicionarmos duas TextView, uma EditText, um Spinner e dois Button. O Código-fonte "SharedPreferences: arquivo actitivity_main.xml" apresenta o arquivo activity_main.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="20dp"
        tools:context=".MainActivity">


    <LinearLayout android:layout_width="match_parent"
                  android:layout_height="match_parent"
                  android:orientation="vertical">

        <TextView android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
                  android:text="Nome"
                  android:textSize="20dp"
                  android:layout_marginTop="20dp"
                  android:layout_gravity="center"/>

        <EditText android:id="@+id/txtNome"
                  android:layout_width="match_parent"
                  android:layout_height="wrap_content"
                  android:layout_marginTop="20dp"/>

        <TextView android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
                  android:text="Tratamento"
                  android:textSize="20dp"
                  android:layout_marginTop="20dp"
                  android:layout_gravity="center"/>

        <Spinner android:layout_width="wrap_content"
                 android:layout_gravity="center"
                 android:layout_height="wrap_content"
                 android:layout_marginTop="30dp"
                 android:scaleX="2"
                 android:scaleY="2"
                 android:drawSelectorOnTop="true"
                 android:entries="@array/saudacoes"
                 android:id="@+id/listTratamento"/>

        <Button android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:layout_marginTop="30dp"
                android:text="Salvar Nova Saudação"
                android:id="@+id/btnSalvar"/>
```

```
        <Button android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:layout_marginTop="30dp"
                android:text="Exibir Saudação"
                android:id="@+id/btnExibir"/>

    </LinearLayout>
</FrameLayout>
```

Código-fonte 5.1 – SharedPreferences: arquivo actitivity_main.xml
Fonte: Elaborado pelo autor (2019)

Uma novidade em relação aos outros exemplos já apresentados até o momento é o Spinner, que consiste em um contêiner de opções e, entre elas, uma única pode ser selecionada pelo usuário. Para implementar o contêiner, é preciso atenção às configurações drawSelectorOnTop (para que o Spinner seja renderizado à frente dos outros widgets) e entries, em que é necessária uma lista de valores para a carga da lista do Spinner. Nesse caso, a lista “saudações” se encontra no arquivo res->values->strings.xml. O Código-fonte “SharedPreferences: arquivo strings.xml” apresenta a lista utilizada nesse exemplo, descrita em um string-array. Com isso, a visualização esperada para a MainActivity é ilustrada na Figura “SharedPreferences – Preview da MainActivity”.

```
<resources>
    <string name="app_name">AppSharedPreferences</string>

    <string-array name="saudacoes">
        <item>Sr</item>
        <item>Sra</item>
        <item>Dr</item>
        <item>Dra</item>
        <item>Sem Tratamento</item>
    </string-array>

</resources>
```

Código-fonte 5.2 – SharedPreferences: arquivo strings.xml
Fonte: Elaborado pelo autor (2019)

Figura 5.1 – SharedPreferences – Preview da MainActivity
Fonte: Elaborado pelo autor (2020)

O botão “Salvar Nova Saudação” salva novos valores para nome e tratamento. Por sua vez, o botão “Exibir saudação” exibe a saudação configurada em outra Activity nomeada SaudacaoActivity (a qual será criada a seguir). O Código-fonte “SharedPreferences: arquivo MainActivity.kt” apresenta a implementação dos listeners de onClick para ambos os botões.

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        btnSalvar.setOnClickListener(View.OnClickListener {

            val saudacaoPersistencia = this.getSharedPreferences("saudacao",
Context.MODE_PRIVATE)
            val editor = saudacaoPersistencia.edit()

            editor.putString("nome",txtNome.text.toString())
            editor.putString("tratamento",listTratamento.selectedItem.toString())
            editor.apply()


            Toast.makeText(this,"Salvo com Sucesso",Toast.LENGTH_SHORT).show()


        })

        btnExibir.setOnClickListener(View.OnClickListener {
```

```
            val intent = Intent(this,SaudacaoActivity::class.java)
            startActivity(intent)


        })

    }

}
```

Código-fonte 5.3 – SharedPreferences: arquivo MainActivity.kt
Fonte: Elaborado pelo autor (2019)

Observando com atenção o uso do SharedPreferences no listener do botão btnSalvar, o método getSharedPreferences entrega uma instância compartilhada desse objeto, de maneira que o desenvolvedor possa configurar o nome do contêiner (nesse caso, chamado de “saudacao”). Por meio desse nome, o desenvolvedor pode acessar novamente os dados gravados em qualquer código-fonte do projeto. O segundo parâmetro define o modo de acesso à área de memória, no qual deve ser definido por padrão o modo privado (MODE_PRIVATE), ou seja, acessado somente pelo aplicativo em execução.

Para adicionar ou editar valores do SharedPrefererentes, é oferecido um objeto associado chamado de Editor, que pode ser acessado pelo método edit(). Sendo assim, o Editor oferece métodos de inserção de valores de diferentes tipos, exigindo o registro de uma chave do tipo String. Neste exemplo, são registradas duas chaves: “nome” e “tratamento” com os dados fornecidos na Activity. Depois, é necessário salvar as inclusões/modificações por meio do método apply().

No botão btnExibir, é instanciada uma Intent para a Activity SaudacaoActivity e, logo em seguida, disparada a transição para a outra Activity por meio do método startActivity().

Para criar outra Activity, o Android Studio possui recursos que facilitam a sua criação, realizando o trabalho de criação do arquivo Kotlin (.kt), arquivo XML para o layout e o registro da Activity no arquivo Manifest. Para isso, basta clicar no botão direito na pasta principal do aplicativo, na aba Project e navegar em New -> Activity -> Empty Activity. Depois, é necessário nomear a Activity e finalizar. Com isso, uma nova activity é criada para edição. As figuras “Caminho para Criação de nova Activity” e “Assistente para criação de nova Activity” ilustram a criação de uma nova Activity pelo assistente do Android Studio.

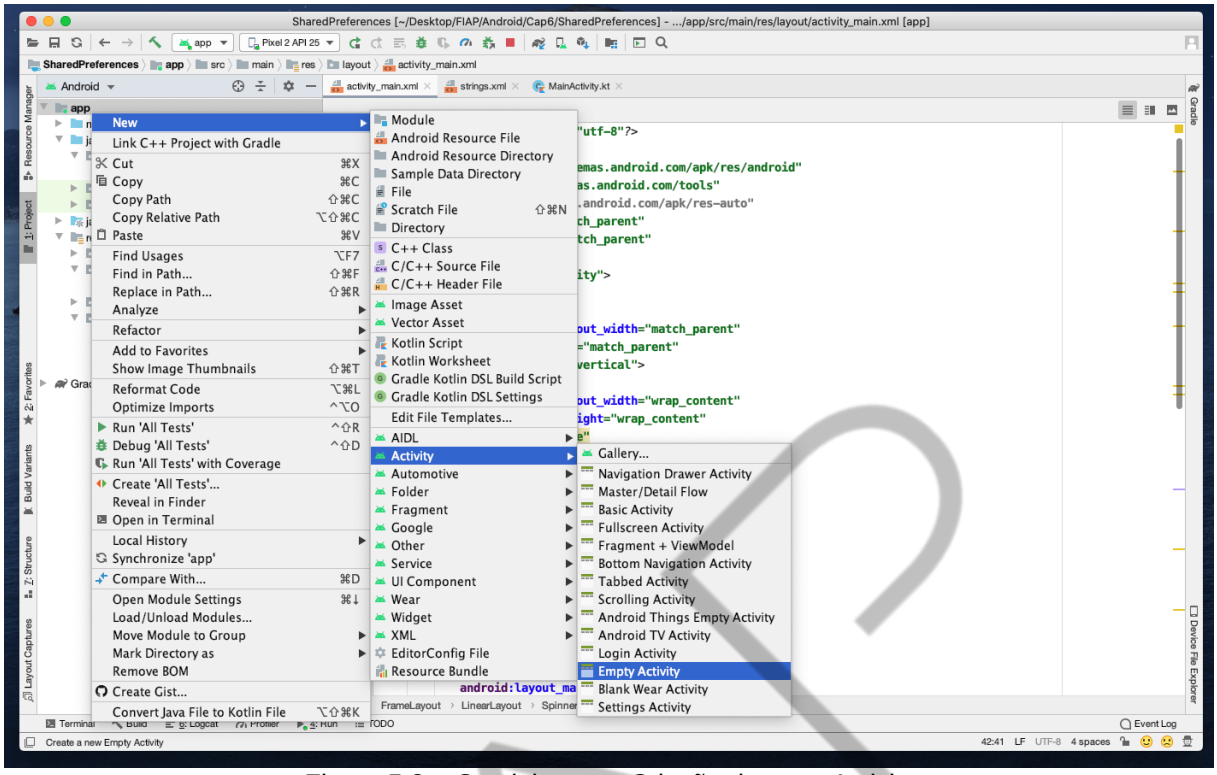

Figura 5.2 – Caminho para Criação de nova Activity
Fonte: Elaborado pelo autor (2020)

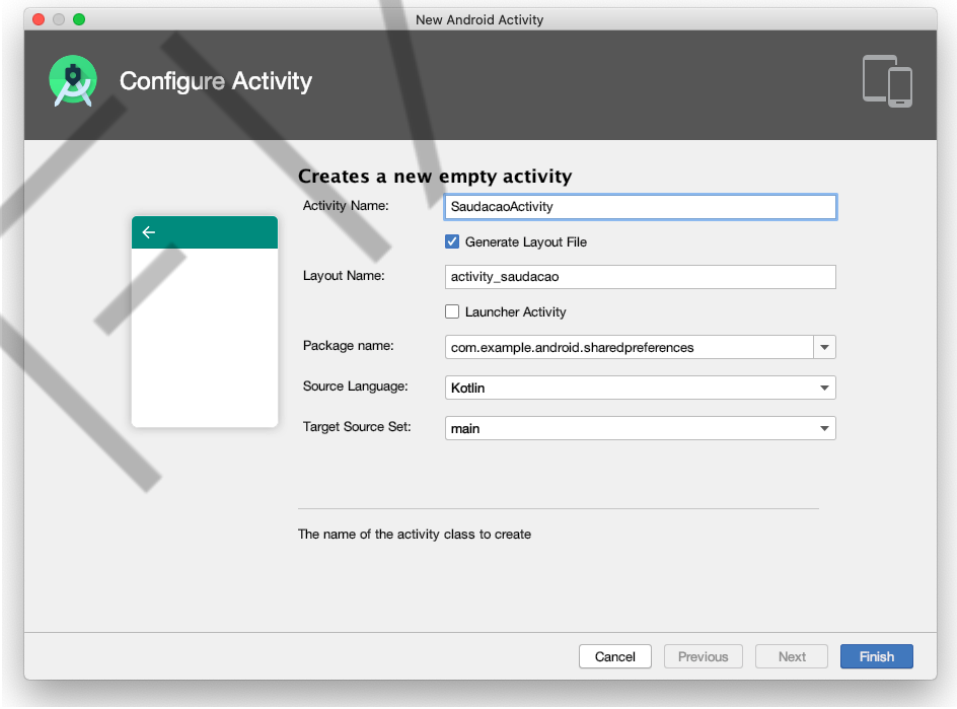

Figura 5.3 – Assistente para criação de nova Activity
Fonte: Elaborado pelo autor (2020)

Depois de criada a Activity SaudacaoActivity, é necessário criar o layout e escrever o código-fonte para acessar os valores de saudação gravados. No layout, foi

inserido um TextView ao centro da tela somente para exibir a saudação gravada. O Código-fonte “SharedPreferences: arquivo activity_saudacao.xml”apresenta o arquivo activity_saudacao.xml criado. A Figura “Preview da saudação_activity.xml” mostra o Preview da activity_saudacao.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="30dp"
        tools:context=".SaudacaoActivity">


    <TextView android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:textAlignment="center"
              android:layout_gravity="center_vertical|center_horizontal"
              android:textSize="20dp"
              android:text="@string/app_name"
              android:id="@+id/lbSaudacao"/>

</FrameLayout>
```

Código-fonte 5.4 – SharedPreferences: arquivo activity_saudacao.xml
Fonte: Elaborado pelo autor (2019)

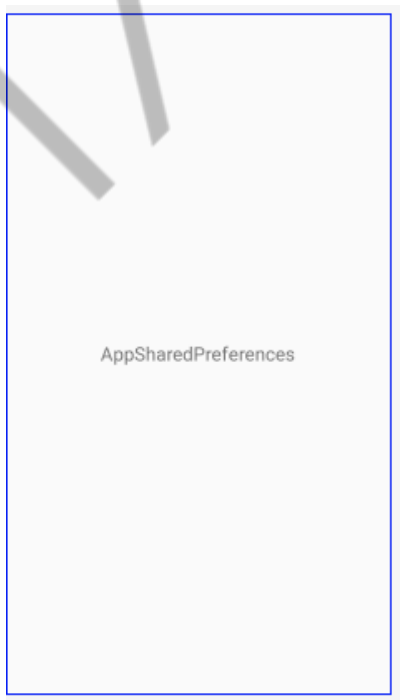

Figura 5.4 – Preview da saudação_activity.xml
Fonte: Elaborado pelo autor (2019)

No código-fonte em Kotlin (arquivo SaudacaoActivity.kt), o trabalho foi recuperar os dados persistidos em memória e exibir na TextView. É possível observar que para recuperar os valores por meio do SharedPreferenfes, basta obter a sua instância compartilhada e acessar os métodos getters, passando como parâmetro a chave desejada e o valor de redundância, caso não seja encontrado nenhum dado na chave (ou também caso a chave não seja encontrada).

No preenchimento do TextView, é feito um tratamento para o valor “Sem tratamento”, no qual deve ser exibido na tela somente o nome informado. Caso contrário, é exibido o tratamento concatenado com o nome. O Código-fonte “SharedPreferences: arquivo SaudacaoActivity.kt” mostra a implementação em Kotlin realizada.

```
class SaudacaoActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_saudacao)

        val saudacaoPersistencia = this.getSharedPreferences("saudacao", Context.MODE_PRIVATE)

        val nome = saudacaoPersistencia.getString("nome","")
        val tratamento = saudacaoPersistencia.getString("tratamento","")

        if(tratamento.equals("Sem Tratamento")){
            lbSaudacao.text = nome
        }
        else{
            lbSaudacao.text = tratamento + " " + nome
        }

    }
}
```

Código-fonte 5.5 – SharedPreferences: arquivo SaudacaoActivity.kt
Fonte: Elaborado pelo autor (2019)

O último passo é adicionar, no arquivo AndroidManifest.xml, a MainActivity como activity pai de SaudacaoActivity por meio da tag parentActivityName. Essa configuração permite que o Android controle a navegabilidade entre as activities automaticamente. O Código-fonte “SharedPreferences: arquivo AndroidManifest.xml” mostra a configuração do Manifest.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="br.com.example.appsharedpreferences">

    <application
            android:allowBackup="true"
            android:icon="@mipmap/ic_launcher"
            android:label="@string/app_name"
            android:roundIcon="@mipmap/ic_launcher_round"
```

```
            android:supportsRtl="true"
            android:theme="@style/AppTheme">
        <activity android:name=".SaudacaoActivity"
                  android:parentActivityName=".MainActivity">
        </activity>
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Código-fonte 5.6 – SharedPreferences: arquivo AndroidManifest.xml
Fonte: Elaborado pelo autor (2019)

O comportamento esperado pelo aplicativo é apresentado nas Figuras “Tela inicial do aplicativo de persistência” e “Tela de saudação do aplicativo de persistência”. Um teste que pode ser feito para validar a persistência do dado no dispositivo móvel é encerrar a aplicação, iniciar novamente e, diretamente, clicar em “Exibir Saudação”. Partindo do pressuposto de que os dados estão gravados na memória, eles serão recuperados.

Esses dados, por sua vez, podem ser limpos em código-fonte por meio do método clear() ou também, com a intervenção do usuário, limpando os dados do aplicativo nas configurações gerais do Android.

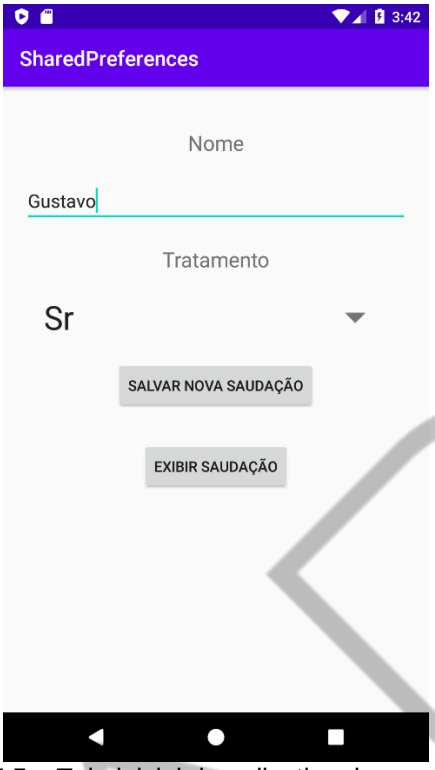

Figura 5.5 – Tela inicial do aplicativo de persistência
Fonte: Elaborado pelo autor (2020)

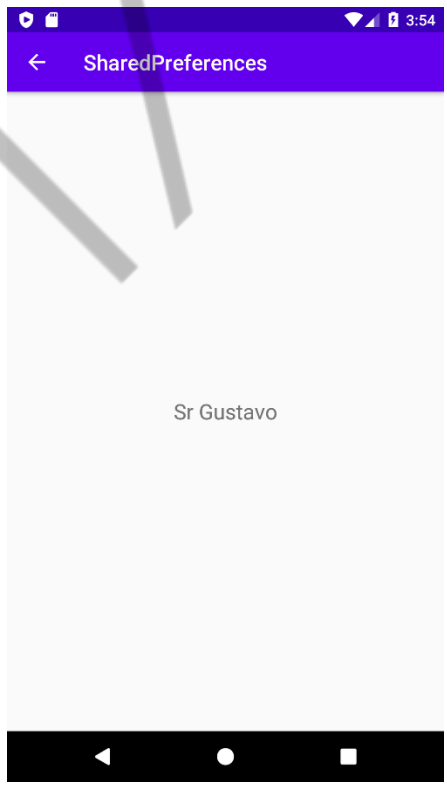

Figura 5.6 – Tela de saudação do aplicativo de persistência
Fonte: Elaborado pelo autor (2020)

5.2 Arquivo

Outra maneira para persistir dados é por meio da gravação e recuperação em arquivo. Para isso, são utilizadas as classes FileOutputStream (para gravação em arquivo) e FileInputStream (para recuperação de dados em arquivo). Para desenvolver o exemplo com arquivos, serão utilizadas as mesmas Activities criadas anteriormente.

Para gravar e recuperar dados no arquivo, foram criadas funções específicas intituladas gravaDadoArquivo e recuperaDadoArquivo no código-fonte do arquivo MainActivity.kt e SaudacaoActivity.kt, respectivamente. O Código-fonte “Arquivo MainActivity.kt com gravação de dado em arquivo” apresenta a MainActivity.kt com a função gravaDadoArquivo implementada e adaptada para salvar as configurações de saudação.

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        btnSalvar.setOnClickListener(View.OnClickListener {

            /*var saudacaoPersistencia = this.getSharedPreferences("saudacao",
Context.MODE_PRIVATE)
            var editor = saudacaoPersistencia.edit()

            editor.putString("nome",txtNome.text.toString())
            editor.putString("tratamento",listTratamento.selectedItem.toString())
            editor.apply()*/


            val data = txtNome.text.toString() + ":" +
listTratamento.selectedItem.toString()
            gravaDadoArquivo("saudacao",data)


            Toast.makeText(this,"Salvo com Sucesso",Toast.LENGTH_SHORT).show()


        })

        btnExibir.setOnClickListener(View.OnClickListener {


            val intent = Intent(this,SaudacaoActivity::class.java)
            startActivity(intent)


        })

    }
```

```
    fun gravaDadoArquivo(filename: String, data: String) {

        try {
            val fs = openFileOutput(filename,Context.MODE_PRIVATE);

            fs.write(data.toByteArray())
            fs.close()
        }
        catch (e: FileNotFoundException){
            Log.i("gravaDadoArquivo","FileNotFoundException")
        }
        catch (e: IOException){
            Log.i("gravaDadoArquivo","IOException")
        }
    }


}
```

Código-fonte 5.7 – Arquivo MainActivity.kt com gravação de dado em arquivo
Fonte: Elaborado pelo autor (2019)

A função gravaDadoArquivo abre o arquivo com o método openFileOutput, passando como parâmetro o nome do arquivo e o modo de escrita (nesse caso, MODE_PRIVATE, para que somente a aplicação tenha acesso a ele). Depois, o conteúdo em texto (com um token ":" utilizado para separar tratamento e nome) é gravado no arquivo, que, então, é fechado. Importante ressaltar que a gravação em arquivo deve ser feita em um ByteArray. A codificação padrão para gravação é em UTF-8. Entretanto, outras codificações de texto podem ser utilizadas, passando a opção como parâmetro do método toByteArray(). O Código-fonte "Arquivo SaudacaoActivity.kt com recuperação de dado em arquivo" mostra a recuperação do dado no arquivo "saudacao", no código-fonte SaudacaoActivity.ky.

```
class SaudacaoActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_saudacao)

        /*var saudacaoPersistencia = this.getSharedPreferences("saudacao",
Context.MODE_PRIVATE)

        var nome = saudacaoPersistencia.getString("nome","")
        var tratamento = saudacaoPersistencia.getString("tratamento","")*/

        val data = recuperaDadoArquivo("saudacao")

        val tokenizer = StringTokenizer(data,":")
        val nome = if (tokenizer.hasMoreTokens()) tokenizer.nextToken() else "Sem
nome"

        val tratamento = if (tokenizer.hasMoreTokens()) tokenizer.nextToken()
else "Sem Tratamento"

        if(tratamento.equals("Sem Tratamento")){
```

```
            lbSaudacao.text = nome
        }
        else{
            lbSaudacao.text = tratamento + " " + nome
        }

    }

    fun recuperaDadoArquivo(filename: String): String {

        try{

            val fi = openFileInput(filename)
            val data = fi.readBytes()

            fi.close()

            data.toString()

            return data.toString(Charset.defaultCharset())
        }
        catch (e: FileNotFoundException){
            return ""
        }
        catch (e: IOException){
            return ""
        }


    }
}
```

Código-fonte 5.8 – Arquivo SaudacaoActivity.kt com recuperação de dado em arquivo
Fonte: Elaborado pelo autor (2019)

Para a recuperação do dado, foi utilizado o método openFileInput, para acessar o dado gravado e extrair, por meio do método readBytes() que entrega os dados em byteArray. Em seguida, o método toString() faz a conversão para o tipo String, lembrando que, nesse caso, é necessário informar a codificação para a conversão (Charset.defaultCharset()). Depois, para tratar o dado recebido, é utilizada a classe StringTokenizer para realizar o parsing do dado, indicando o caractere ":" como token utilizado. Com isso, o método nextToken() consegue entregar separadamente nome e tratamento. Espera-se o mesmo comportamento do exemplo anterior, entretanto, com o uso de arquivo.

5.3 SQLite

A terceira maneira pela qual o dado pode também ser persistido em memória é por meio do uso do SQLite nativo, disponível no Android. Para viabilizar o uso, é

necessário criar uma classe que gerencie a base de dados. O Código-fonte “Classe DatabaseManager” mostra a classe DatabaseManager criada, na qual gerencia a criação e a manipulação dos dados. Será criada uma tabela chamada SAUDACAO com os campos ID_SAUDACAO, NOME e TRATAMENTO.

Para criar a classe em Kotlin, clique com o botão direito na pasta *java*, dentro da janela de Projeto do nosso aplicativo; depois, selecione a opção *New* e, por último, selecione a opção do *menu Kotlin File/Class,* como mostra a Figura “Criação de classe em Kotlin”.

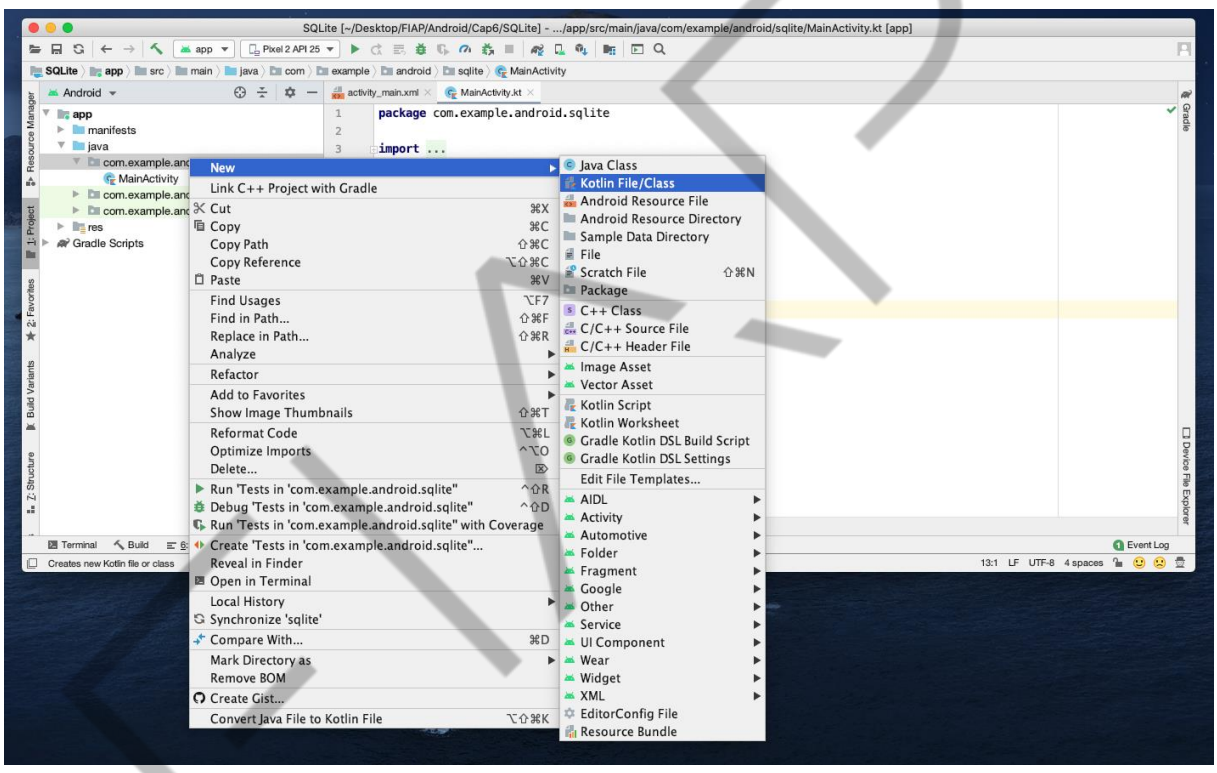

Figura 5.7 – Criação de classe em Kotlin
Fonte: Elaborado pelo autor (2020)

No Assistente exibido, vamos colocar o nome do nosso arquivo e dizer que ele vai ser do tipo *Class*, como mostra a Figura “Assistente de criação de arquivo”.

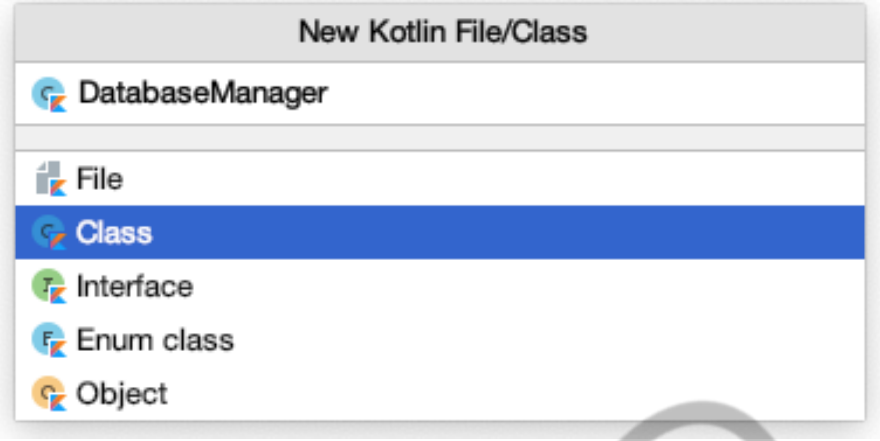

Figura 5.8 – Assistente de criação de arquivo
Fonte: Elaborado pelo autor (2020)

```
class DatabaseManager(context: Context, name: String?) : SQLiteOpenHelper(context,name,null,1)
{



    override fun onCreate(p0: SQLiteDatabase?) {

        p0?.execSQL("CREATE TABLE SAUDACAO(\n" +
                "\tID_SAUDACAO INT NOT NULL,\n" +
                "\tNOME VARCHAR(20),\n" +
                "\tTRATAMENTO VARCHAR(20),\n" +
                "\tPRIMARY KEY (ID_SAUDACAO)\n" +
                "\t);")
    }

    override fun onUpgrade(p0: SQLiteDatabase?, p1: Int, p2: Int) {
        p0?.execSQL("DROP TABLE IF EXISTS SAUDACAO")

        p0?.execSQL("CREATE TABLE SAUDACAO(\n" +
                "\tID_SAUDACAO INT NOT NULL,\n" +
                "\tNOME VARCHAR(20),\n" +
                "\tTRATAMENTO VARCHAR(20),\n" +
                "\tPRIMARY KEY (ID_SAUDACAO)\n" +
                "\t);")
    }

    fun insereSaudacao(id: Int, nome: String, tratamento: String){
        var db = this.writableDatabase

        var cv = ContentValues()

        cv.put("ID_SAUDACAO",id)
        cv.put("NOME",nome)
        cv.put("TRATAMENTO",tratamento)

        db.insert("SAUDACAO","ID_SAUDACAO",cv)
    }

    fun listaSaudacao(): Cursor {

        var db = this.readableDatabase
        var cur = db.rawQuery("select nome,tratamento from saudacao",null)
        return cur
    }

    fun removeSaudacao(){
        var db = this.writableDatabase
        db.delete("SAUDACAO","ID_SAUDACAO=1",null)
    }
```

```
}
```

Código-fonte 5.9 – Classe DatabaseManager
Fonte: Elaborado pelo autor (2019)

Primeiramente, a classe dedicada ao gerenciamento do banco de dados deve herdar da classe SQLiteOpenHelper e, por sua vez, implementar, necessariamente, dois métodos chamados onCreate (faz a criação do banco de dados) e onUpgrade (atualiza a estrutura do banco de dados para casos de mudanças de versão). Para o exemplo que segue, será mantida uma única versão.

Observe que o método onCreate faz a criação da tabela por meio do comando SQL CREATE. Para inserções de dados, um objeto assistente chamado ContentValues faz assistência à organização do registro para inserção.

Para pesquisas, o comando SQL SELECT pode ser executado e a tabela-resultado é armazenada em um objeto Cursor. Por sua vez, para remoção de registros, deve ser utilizado o comando delete. Os códigos-fonte “Uso do DatabaseManager na classe MainActivity” e “Uso do DatabaseManager na classe SaudacaoActivity” mostram o uso do DatabaseManager na classe MainActivity e SaudacaoActivity.

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val db = DatabaseManager(this,"saudacoes")

        btnSalvar.setOnClickListener(View.OnClickListener {

            /*var saudacaoPersistencia = this.getSharedPreferences("saudacao",
Context.MODE_PRIVATE)
            var editor = saudacaoPersistencia.edit()

            editor.putString("nome",txtNome.text.toString())
            editor.putString("tratamento",listTratamento.selectedItem.toString())
            editor.apply()*/


            /*var data = txtNome.text.toString() + ":" +
listTratamento.selectedItem.toString()
            gravaDadoArquivo("saudacao",data)*/

            db.removeSaudacao()
            db.insereSaudacao(1,
txtNome.text.toString(),listTratamento.selectedItem.toString())

            Toast.makeText(this,"Salvo com Sucesso",Toast.LENGTH_SHORT).show()


        })
```

```
        btnExibir.setOnClickListener(View.OnClickListener {


            val intent = Intent(this,SaudacaoActivity::class.java)
            startActivity(intent)


        })

    }
}
```

Código-fonte 5.10 – Uso do DatabaseManager na classe MainActivity
Fonte: Elaborado pelo autor (2019)

```
class SaudacaoActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_saudacao)

        val db = DatabaseManager(this,"saudacoes")

        /*var saudacaoPersistencia = this.getSharedPreferences("saudacao",
Context.MODE_PRIVATE)

        var nome = saudacaoPersistencia.getString("nome","")
        var tratamento = saudacaoPersistencia.getString("tratamento","")*/

        /*var data = recuperaDadoArquivo("saudacao")
        var tokenizer = StringTokenizer(data,":")
        var nome = tokenizer.nextToken()
        var tratamento = tokenizer.nextToken()*/

        val cursor = db.listaSaudacao()

        var nome = ""
        var tratamento = ""

        if(cursor.count > 0){
            cursor.moveToFirst()

            nome = cursor.getString(cursor.getColumnIndex("NOME"))
            tratamento = cursor.getString(cursor.getColumnIndex("TRATAMENTO"))
        }


        if(tratamento.equals("Sem Tratamento")){
            lbSaudacao.text = nome
        }
        else{
            lbSaudacao.text = tratamento + " " + nome
        }

    }

    fun recuperaDadoArquivo(filename: String): String {

        try{

            val fi = openFileInput(filename)
            val data = fi.readBytes()

            fi.close()

            data.toString()

            return data.toString(Charset.defaultCharset())
```

```
        }
        catch (e: FileNotFoundException){
            return ""
        }
        catch (e: IOException){
            return ""
        }


    }
}
```

Código-fonte 5.11 – Uso do DatabaseManager na classe SaudacaoActivity
Fonte: Elaborado pelo autor (2019)

Na classe SaudacaoActivity, o resultado da pesquisa do nome e o tratamento para ser exibido devem ser obtidos de um cursor, sendo semelhante ao consumo de coleções de dados (por exemplo, ArrayList ou ResultSet). Caso o cursor tenha resultados, ele pode ser percorrido e os dados dos registros obtidos por seus respectivos getters e indexadores de campos.

CONCLUSÃO

A plataforma de desenvolvimento de aplicativos Android permite ao desenvolvedor diferentes maneiras de persistir dados, utilizando diferentes soluções: cada uma com sua especificidade e aderência ao problema proposto.

A persistência de dados locais também auxilia em situações quando é necessário salvar os dados — mesmo sem ter uma conexão de dados estável — para, posteriormente, sincronizar os dados de maneira remota.

REFERÊNCIAS

ANDROID. **Room Persistence Library**. 2018. Disponível em: <https://developer.android.com/topic/libraries/architecture/room>. Acesso em: 06 out. 2020.

ANDROID. **SharedPreferences**. 2020. Disponível em: <https://developer.android.com/reference/android/content/SharedPreferences>. Acesso em: 06 out. 2020.